ITOS Page & Seqprt Definition Guide

Integrated Test & Operations System
User's Guides $Date: 2006/03/21 16:07:24 $

ITOS Development & Support Group
NASA/GSFC Code 584, Greenbelt MD 20771

1 Introduction

The ITOS provides four broad capabilities for displaying telemetry mnemonic values: display pages, sequential prints, plots and stripcharts.

1. There are two types of display pages: text and graphic. Text pages are described in this document. Graphic pages can contain text, gauges and plots and are described in the Graphic Page User's Guide.
2. Sequential prints consist of plain text and information about telemetry mnemonics, principly the mnemonic's value. They are also described in this document.
3. Plots are described in the ITOS Plotting User's Guide.
4. Stripcharts are described in the STRIPCHART directive.

The display page is a color window on the display which shows mnemonic values and other information. Each item – pure text, mnemonic information, or a mixture – is positioned at a certain character row and column on the page. Mnemonic information is updated on a timed interval. Page displays may be duplicated to the system's general-purpose printer or to a file using the STOL `snap` directive or the page's pop-up menu. Snap pages are not in color and mnemonic values are updated at the time the snap is performed (so it might not look exactly like the displayed page).

The sequential print captures a set of mnemonic values. Each text or mnemonic item is positioned beginning at a certain column on the print page, and all values are printed in one row. Column headings may be supplied. Values may be updated as they are received by the TCW system, or on a timed interval. Prints may be directed to the display, a file, the general-purpose printer, the event queue, or any combination of these.

Telemetry display pages and sequential prints are defined in disk files using simple definition languages. Each page or print must be defined in a separate file, where the file name is the page name plus the apropriate suffix, as follows:

'`.page`' – identifying the file as a display page definition; or
'`.sprt`' – identifying the file as a sequential print definition.

Display programs locate page and sequential print definition files by searching the directories listed in the global mnemonic `GBL_PAGEPATH`. This string mnemonic contains a list of directory pathnames separated by colons. If the environment variable `ITOS_PAGEPATH` is set when the system is started, the global is set from that environment variable. **File names must be in lower case** for the search to succeed.

The definition scheme for pages and prints is nearly identical. The principal difference is that page items may be placed in two dimensions – row, column – while seqprt items may be placed in only one dimension – column. Some statements are useful only for one or the other; these will be pointed out in the following text.

2 Telemetry Page & Seqprt Definition Files

Page definitions mainly consist of statements telling the system where on the page to place information about various mnemonics. They also may contain statements controlling the coloration of items on the page, statements supplying mnemonic value conversions, and statements describing the page's purpose. Let's begin with a version of the ITOS status page as an example:

```
page status

disc "Default TCW system status page."

discrete retr (
    OFF (0,0,black,yellow)
    ON  (1,100, black, green)
)

color (
    banner (blue,default)
    cmd_mode (blue,default)
    rpt_stat (blue,default )
)

banner       (  1, 13, "SYSTEM STATUS" )
GBL_GMTOFF   (  +,  1, ":value%D: GMT" )
GBL_DBVERS   (  =, +4, "DB Vers:: :value%5.2f:")
GBL_ACQUIRE  (  +,  1, "Acquire:: :value%-32.32s:" )
GBL_TPKTAPID (  +,  1, "Last Pkt ID:: :value%4d:" )
gbl_totmpkts (  =,  23, "# Pkts Rcvd:: :value%6d:")
gbl_tlmrate  (  +,  1, "Simulated Tlm Rate:: :value%4.2f:" )

cmd_mode        (+1, 1, "Cmd Mode")
rpt_stat        (=, 23, "Cmd Report Status")
GBL_CMD_MODE    (  +,  1, "Mode:: :value%3.3s:STEP" )
GBL_CLCW_RV_0   (  =, 23, "CLCW RV:: :value%3.3d:" )
GBL_RETRLIM_0   (  +,  1, "Auto Retry::   :value%3.3s:", retr )
GBL_CLCW_LCK_0  (  =, 23, "CLCW Lockout::  :value%3.3s:" )
GBL_RETRLIM_0   (  +,  1, "Num Retries::  :value%2d:" )
GBL_CLCW_RET_0  (  =, 23, "CLCW Retxmit::  :value%3.3s:" )
GBL_BP_MODE     (  +,  1, "Bypass Mode::  :value%3.3s:" )
GBL_CLCW_FMB_0  (  =, 23, "CLCW Farm B:: :value%4d:" )
GBL_FOPTIMEOUT_0(  +,  1, "CLCW Timeout:: :value%2d: sec" )
GBL_FOPSTATE_0  (  =, 23, "FOP State:: :value%10.10s:")
```

Every page definition must begin with a page statement, and must contain at least one mnemonic or text statement. All other statements are optional.

Seqprt definitions are very similar. They mainly consist of statements telling the system in what column to place information about mnemonics. They don't support color, but do support conversions, and there are several statements specific to seqprt definitions. Here is a short example:

```
seqprt agyro

noholes
lastline 200000000

heading h034time  (1,  1, ":n:")
heading adp_gyrtx (1, +2, ":n:")
heading adp_gyrty (1, +2, "ACGYRORATEY")
heading adp_gyrtz (1, +2, "ACGYRORATEZ")

h034time  ( (1, ":v%10[%10d]C:",raw) (+,".:v%5[%f]D:") )
adp_gyrtx ( +2, ":v%10.7f:")
adp_gyrty ( +2, ":v%10.7f:")
adp_gyrtz ( +2, ":v%10.7f:")
```

Each seqprt definition should begin with a `seqprt` statement, and must contain at least one mnemonic or text statement.

2.1 Mnemonic Statement

Syntax

mnemonic (*row*, *column*, *format*[, *conversion*])
mnemonic ((*row*, *column*, *format*[, *conversion*])[,] ...)

Page definition mnemonic statements.

mnemonic (*column*, *format*[, *conversion*])
mnemonic ((*column*, *format* [, *conversion*])[,] ...)

Sequential print definition mnemonic statements.

where the variables are:

mnemonic is a telemetry mnemonic.

row (**pages only**) is the page row, in characters, at which to begin displaying this item.

column is the page column, in characters, at which to begin displaying this item.

format details what to display and how.

conversion gives the how to convert the value being displayed. This is the name of a conversion defined earlier in the file or one of the following keywords:

`raw` Do not display the mnemonic's converted value regardless of the state of it's operational database `convert` flag.

`nat` Display the mnemonic's value in the way most natural for the GSE computer. For example, FAST spacecraft BCD floating-point values would be displayed in floating-point notation, whereas the `raw` keyword would produce the hexidecimal number received by the GSE.

`cnv` Display the converted mnemonic's value if there is a conversion associated with the mnemonic in the database and the mnemonic's `convert` flag is true. If there is not database conversion, the natural form will be displayed as if `nat` had been specified. Otherwise the raw value will be displayed.

Examples

```
pbatvolt (1, 1, ":name:  :value%5.2f:  :units: :static::quality::limits:")
pbatcurr (+, =, ":n:  :v%4d:  :u: :s::q::l:", raw)
prelay1  (+, =, ":n:  :v%3s:       :s::q::l:", relay)
moredata ( (1, 32, ":n:") (+, =, "    :v%4s:") )
```

Discussion

To place mnemonic information on a page or seqprt, use the mnemonic statement. This statement begins with a mnemonic name and is followed by a parenthesized group of instructions on where and what to display about the mnemonic.

In page definitions, inside the parentheses are four things, in order: a row position, a column position, a format string, and an optional conversion tag. Row and column position are given in characters. The standard page is 80 characters across by 18 rows.

In seqprt definitions, there is no row position so there are only three things inside the parenthesis: the row position, format string and conversion tag.

The format is discussed in Chapter 5 [Format Strings], page 19.

The conversion tag may be either one of three words, given above, with special meaning to the page system, or the name of a conversion previously defined in the same page definition file. Section 2.7 [Analog Statement], page 6 and Section 2.8 [Discrete Statement], page 7 for details. If this field is not given, it defaults to `cnv`.

2.2 Text Page Statement

Syntax

[*name*] (*row*, *column*, *string*)
[*name*] ((*row*, *column*, *string*) [,] ...)

Page definition text item statements.

[*name*] (*column*, *string*)
[*name*] ((*column*, *string*)) [,] ...)

Sequential print definitions text item statements.

Discussion

The text statement describes where a given text string should be displayed.

Like the mnemonic statement, the *row* and *column* parameters give the starting character location for the text string. As shown above, no row position may be specified in seqprt definitions.

The quoted *string* parameter is the text to be displayed at the given location. Newline characters are not honored in text strings.

Text statements are of limited use in seqprt definitions.

2.3 Definition File Comments

Syntax

`#` *text*

Discussion

Page and seqprt definition files may contain comments anywhere within them. Comments begin with the '`#`' character and continue to the end of the line.

The comment character inside a quoted string is not taken as the beginning of a comment, but is included as is in the string. You also may use '`\#`' to put a literal pound sign in a page definition.

2.4 Include Statement

Syntax

`include` *filename*

Discussion

The include statement provides a means of including partial page definitions in a page definition. Files to be included must be in a directory in the page search path. Included files should not contain `page` statements. Include files may be nested up to nine deep.

2.5 Description Statement

Syntax

`desc` *text*

Discussion

The description statement provides a means of including text describing a page definition. Users may be able to view page descriptions through ITOS in a later release.

Multiple description statements are concatenated together to form one continuous description text.

2.6 Debug Statement

Syntax

`yydebug`

Discussion

The debug statement provides a means of turning on page parser debugging facilities. It is intended for use by the ITOS developer staff and never should be included in an operational page definition.

2.7 Analog Statement

Syntax

`analog` *name* `(`*coef_list*`)`

Define an on-page analog conversion.

`analog ( `*analog_stmnt*`[,] ... )`

Define a group of analog conversions.

where the variables are:

name is the analog conversion name used in mnemonic statements.

coef_list is a list of polynomial coeficients.

analog_stmnt
is an analog statement without the `analog` keyword.

Discussion

The analog statement defines a polynomial for an analog conversion which may be applied one or more mnemonics on the page.

Each analog conversion consists of a *name* followed by a list of coefficients. The *name* is the name by which the conversion may be referenced in mnemonic statements. See Section 2.1 [Mnemonic Statement], page 3.

The coefficient list is a comma-separated list of up to 8 numbers representing coefficients for $x\hat{}0$ through $x\hat{}7$. Unspecified coefficients are set to zero.

2.8 Discrete Statement

Syntax

`discrete` *name* `(`*text* `(` *low*`,` *high*`,` *fg*`,` *bg* `)[,] ...)`

Define an on-page discrete conversion.

`discrete (` *discrete_stmnt*[`,`] ... `)`

Define a group of discrete conversions.

where the variables are:

name is the discrete conversion name used in mnemonic statements.

text is the text to display if the value falls in this state.

low is the low value of the discrete range.

high is the high value of the discrete range.

fg is the foreground color in which to display the *text*.

bg is the background color in which to display the *text*.

discrete_stmnt
is an discrete statement without the `analog` keyword.

Discussion

The discrete statement defines a discrete conversion which may be applied to any mnemonic on the page.

Each discrete conversion consists of a *name* followed by a list of discrete states. The *name* is the name by which the conversion may be referenced in mnemonic statements.

Each discrete state consists of *text* and a parameter list. The *low* and *high* parameters give a range of mnemonic values for which the given text should be displayed. The *fg* and *bg* parameters give the colors in which the text should be displayed.

2.9 General Rules for Definition Files

The page definition format is not line oriented. Spaces, tabs, and newline characters are all treated as *whitespace*. Extraneous whitespace is ignored. Multiple item descriptions may occur on the same line and item descriptions may be split over several lines.

In mnemonic and text statements, the *row* and *column* parameters may contain the symbols `+`, `-`, and `=`. Row or column numbers preceded by a `+` are replaced by the given number plus the ending row or column of the preceding item in the definition. Row or column numbers preceded by a `-` are replaced by the given number minus the starting row or column of the preceding item in the definition. If `+` or `-` appear alone, a one is assumed. (Note: Items may only contain one row, so starting and ending rows are always the same.) The symbol `=` may be used to specify that the row or column should be the same as the starting row or column of the preceding item.

For sequential prints, printing will begin with the first heading line and end with printing the line specified in the lastline statement. If no headings are defined, printing will begin on line one. If no lastline is specified, the last line will be line 66.

Syntatic Rules

- Statement identifier keywords are `page`, `seqprt`, `include`, `desc`, `debug`, `discrete`, and `analog`. Additional keywords for page definitions are `button` and `color`. Additional keywords for sequential print definitions are `heading`, `lastline`, `partial`, `frequency`, and `trigger`. Following the keyword `color`, `default` and `mnedef` are keywords. Keywords may not be used as text or mnemonic identifiers.
- In mnemonic color statements, keywords are `name`, `value`, `units`, `static`, `quality`, `limits`, and `text` for mnemonic data parts.
- In mnemonic statements, keywords are `cnv`, `nat`, and `raw` for the *conversion* parameter. These may not be used for conversion names.
- Parameters all applying to the same item or key word must be grouped with parenthesis. For example, all parameters applied to a display item must be collected in parenthesis: `tlmint8(1,1, ":n: :v%8d:")`. It is OK to write `discrete disc1(...) discrete disc2(...)` or `discrete(disc1(...)disc2(...))`.
- Items are delimited by whitespace, except in a parenthetical list, where they are separated by commas. Where parenthesis are allowed or required, they may serve as delimiters instead of or in addition to whitespace and commas. Extraneous whitespace is ignored.

  For example:

  ```
  discrete(disc1(...), disc2(...)) analog anlg1(...)
  discrete(disc1(...)disc2(...))analog(anlg1(...)
  ```

- Multi-word strings must be enclosed in quotes. Single-word strings need not be quoted. In strings, characters following a backslash (\) are taken literally and the backslash itself is dropped. So, `\"` is a quote character in a string. Keywords may appear in quoted strings. Quoted strings may not contain newwline characters.

Examples

```
page example
desc "This is an example of a page definition"

color default ( 1, blue )               # one way to do colors
color                                   # the other way to do colors
(
    text_a     ( 1, red )               # text_a colors
    mnedef                              # mnemonic defaults
    (
        name  ( 2, blue )
        value  ( yellow, 1 )
        units  ( 3, green )
    )
    tlmint8                             # tlmint8 colors
    (
        name  ( 4, 5 )
        value  ( 2, yellow )
        units  ( red, green )
    )
)

discrete disc1                         # discrete values for tlmint16
(
#-------------------------------------------------------------------
# string      low     high     fg       bg
#-------------------------------------------------------------------
   OFF       (  1,       1,      3,      NavajoWhite )   # OFF = 1
   ON        (  2,       2,      1,      =           )   # ON  = 2
)

analog                                 # not used in this example
(
#-------------------------------------------------------------------
#               0      1     2     3     4
#-------------------------------------------------------------------
    analog1 (  3.2,   4.3,   0,   4.0       ),
    analogb (  2.14,  5.0,   0,     0,  2.39)
)

#-------------------------------------------------------------------
# mnemonic     row    col    fmt                                 cnv
#-------------------------------------------------------------------
  tlmint8  ((   1,     1,    "Integer mnemonic :v%8d:",       raw  ),
           (   =,     +2,    ":v%10.3f: :u:",                 cnv  ))
  tlmstr12  (   +,     =,    "String mnemonic  :v%-64.64s:",  cnv  )
  tlmflt32  (   =,     +2,    ":v%8.2f:",                     cnv  )
  text_a    (  +2,     10,    "This is just raw text"              )
            (   +,     =,    "More raw text on the next line"     )
  tlmint16  (  +2,    -10,    ":value%4s:",                   disc1)
```

```
page example
desc "This is the same as the above example without"
desc "as pleasant an appearance"
color default(1, blue)
color(text_a( 1, red)
      mnedef(name(2, blue)value(yellow, 1)units(3, green))
      tlmint8(name(4, 5)value(2, yellow)units(7, green)))
discrete disc1(OFF(1, 1, 3, NavajoWhite)        # OFF = 1
               ON(2, 2, 1, = ))                 # ON  = 2
analog(analog1(3.2, 4.3, 0, 4.0),
       analogb(2.14, 5.0, 0, 0, 2.39))
tlmint8((1, 1, "Integer mnemonic :v%8d:", raw),
        (=, +2, ":v%10.3f: :u:", cnv))
tlmstr12(+, =, "String mnemonic  :v%-64.64s:", cnv)
tlmflt32(=, +2, ":v%8.2f:", cnv)
text_a(+2, 10, "This is just raw text")
(+, =, "This is more raw text on the next line")
tlmint16(+2, -10, ":value%4s:", disc1)
```

```
#-------------------------------------------------------------------------
# Sample sequential print page
#-------------------------------------------------------------------------
#
# This is the sequential print definition for printing seds real-time
# events.
#
# Note that headings are automatically suppressed by the sequential
# print software when printing to the event log.

seqprt events

desc "Print SAMPEX real-time event messages"

heading  (
         ( 3,  1, "Pkt Time"       )    # 2-line page head margin
p004time ( +,  =, ":name:"         ))
p004time (     =, ":value%T:"      )    # packet 4 time
heading  (
         ( -, 22, "Pkt 4 Count"    )
p004cnt  ( +,  =, ":name:"         ))
p004cnt  (     =, ":value%6d:"     )    # packet 4 count
heading  (
         ( -, 36, "Event Pkt"      )
srtevent ( +,  =, ":name:"         ))
srtevent (     =, ":value%-64.64s:")    # event message text

heading  (5, 1, ""                 )    # blank line under headings

lastline 63                             # 3-line page foot margin
```

3 Statements For Page Definitions Only

There are three statements which may appear only in page definition files: the `page` statement, naturally, which begins a page definition; the `color` statement, because sequential prints do not support color output; and the `button` statement, because buttons on sequential prints don't really make sense.

The `color` statement controls the foreground and background colors of items on a page. By default, all items appear as white text on a blue background. This can be changed globally or item-by-item with the `color` statement.

The `button` statement allow you to put an active push button on a page. The button can be programmed to execute any STOL directive when the button is activated with a mouse click.

3.1 Page Statement

Syntax

`page` *name*

Discussion

Every page definition file must begin with a page statement, giving the page name. The name should match the filename, without the '`.page`' extension. If the names do not match, the system will complain, but the page definition will not be rejected.

Only comments and whitespace may precede the page statement, and only one page statement may appear in a definition file. (Subsequent page statements are reported as unrecognized statements.)

3.2 Color Statement

Syntax

`color default (` *color_pair* `)`

Set default colors for all items.

`color` *text_name* `(` *color_pair* `)`

Set the colors for a given text or button item.

`color mnedef ( [` *field* `(` *color_pair* `) [,] ... ] )`

Set the default colors for all mnemonic items.

`color` *mnemonic* `( [` *field* `(` *color_pair* `) [,] ... ] )`

Set the colors for a given mnemonic item.

`color (` *color_dfn* `[,] ... )`

Collect a bunch of color statements together.

where the variables are:

color_pair is a comma-separated foreground and background color.

text_name is the name assigned to a text statement or button statement.

mnemonic is a mnemonic name used in a mnemonic statement.

field names a bit of information about a mnemonic like it's name or value:

`name`	mnemonic name from the database
`value`	mnemonic value
`units`	mnemonic units-of-measure string from the database
`static`	mnemonic static flag indicator
`quality`	mnemonic quality flag indicator
`limits`	mnemonic limits flag indicator
`text`	any text in the mnemonic format string

Discussion

The color statement describes the colors which should be associated with the various parts of a given item. It consists of the `color` keyword followed by one of four identifying words and appropriate color information. The four identifiers are: the keyword `default`, the keyword `mnedef`, a text item name, or a mnemonic name.

The *color_pair* parameter is a comma-separated pair of colors representing foreground and background colors, in that order. Individual colors may be given as color codes 0 through 7, as described in the Telemetry and Command Handbook, or as color names recognized by the X Window System.

The `color default` statement sets default colors for all text on the page. In the absence of other color statements, this would set the colors for all text and mnemonic items. The `color` *text_name* statement sets colors for the particular text item named.

The `color mnedef` statement sets default colors for all mnemonic items on the page. The `color` *mnemonic* statement sets colors for the particular mnemonic item named.

The mnemonic color statements have parameters representing pieces of information about the mnemonic which may be displayed. Each of these subparts may have different color settings.

3.3 Button Statements

Syntax

`button` [*name*] `(` *row*`,` *column*`,` *text*`,` *directive* `)`
`button` [*name*] `( (` *row*`,` *column*`,` *text*`,` *directive* `)` [`,`] ... `)`

Page definition button statements.

where the arguments are:

name is a name assigned to the button.

row is the page row, in characters, at which to begin displaying this button.

column is the page column, in characters, at which to begin displaying this button.

text is the text with which to label the button.

directive is the text of a STOL directive to execute when the button is clicked.

Discussion

The button statement allows you to put active push buttons on any page. The button is labeled with the given *text* and when the mouse is clicked in the button, the given STOL *directive* is sent to the STOL interpreter for execution.

The directive can be anything; for example: a `page` directive that will bring up a new page, a `start` directive that will start a STOL proceedure, or even a spacecraft command.

Button colors can be controlled as if they are ordinary text items; that is, the page color statements can include button *name*s as well as text item names.

4 Statements for Seqprt Definitions Only

This is the seqprt defs node.

4.1 Seqprt Statement

Syntax

`seqprt` *name*

Discussion

Every sequential print definition file must begin with a seqprt statement, giving the seqprt name. The name should match the filename, without the '`.sprt`' extension. If the names do not match, the system will complain, but the print definition will not be rejected.

Only comments and whitespace may precede the seqprt statement, and only one seqprt statement may appear in a definition file. (Subsequent seqprt statements are reported as unrecognized statements.)

4.2 Lastline Statement

Syntax

`lastline` *number*

Discussion

The lastline statement provides a means of defining the last print line on a sequential print page. The line *number* given in this statement will be the last line on the page on which data is printed before a top-of-form is performed.

4.3 Firstline Statement

Syntax

`firstline` *option*

where *option* may be one of

`on` Prints an output line immediately when the SEQPRT directive is given. Values are obtained from the CVT

`off` (default) means that nothing is output until values are received.

Discussion

The `firstline` statement provides a means of forcing one line of output as soon as the SEQPRT directive is given. By default, no output is printed until the conditions specified by the frequency and trigger statements are met.

When a rate is specified with the SEQPRT directive, the first line of output is always created immediately regardless of the option used with `firstline`.

4.4 Partial Statement

Syntax

`partial` *option*

where *option* may be one of

`ok`	(default) says print lines with blanks
`fill`	says supply missing values from the CVT
`discard`	says drop incomplete lines.

Discussion

The `partial` statement provides a means of controlling how incomplete output lines are handled.

The sequential print process receives values in which it has an interest from the ITOS telemetry subsystem as they arrive at the data unpacker. The sequential print process normally prints a line of output whenever all values for the print have been received or any value is received a second time. See Section 4.6 [Trigger Statement], page 17. If all values to be printed for a given page have not been received before a value is received a second time, by default (`partial ok`) the data will print with blanks where the missing values would be.

This occurs when data comes from different packets which are telemetered at different rates or when data overflows the internal communication channel connecting the sequential print process with the telemetry processes.

If the `partial discard` statement is included in a page definition, such lines with data *holes* are not printed; they are dropped. If `partial fill` is specified, missing data are filled in with the most recent values from the ITOS current value table (CVT).

4.5 Frequency Statement

Syntax

`frequency` *option*

where *option* may be

`set`	(default) ask for mnemonic values whenever they are set.
`changed`	ask for mnemonic values only when they change.

Discussion

Sequential prints can get be set up to get each telemetry value every time the value is received, or only when the value received is different from the previous value. So to potentially reduce the number of values received for printing, include `frequency changed` in the seqprt definition.

4.6 Trigger Statement

Syntax

`triger` *option*

where *option* may be one of

`norm` (default) for normal triggering.

`pkt` for triggering on packet boundaries.

Discussion

By *triggering*, we mean, "when does the seqprt program write a line of output?"

In normal triggering (`norm`), the default, the output line is written when all values have been received, or when any value is received a second time while the output line is being constructed.

That is: The seqprt program collects and stores the values it needs to produce a line of output. When it has a complete set of values, it will write the line and start over. However, if is has not received all the values it needs and it receives a value for a mnemonic for which it already has a value, then it will print the line and start over collecting new values for all the mnemonics. This is one case where the output may have *holes*. See Section 4.4 [Partial Statement], page 16.

Packet (`pkt`) triggering also is subject to producing output with holes. The seqprt program is notified whenever the telemetry unpacker finishes unpacking a packet that contained data wanted by the seqprt. If `pkt` triggering is in effect, the seqprt program will write the output line at that point, regardless of whether the line is complete.

Packet triggering is useful when doing a seqprt on mnemonics from different packets. Since packets often arrive at different rates, it assures that each value in a line is contemporaneous with the other values, since they all came from the same packet.

4.7 Heading Statement

Syntax

`heading` *mnemonic-stmnt*
`heading` *text-stmnt*
`heading ( ` *either-stmnt ...* `)`

Discussion

The heading statement describes where and how a mnemonic-data or where a pure-text heading should be printed on a sequential print page. It consists of the `heading` keyword followed by a page-style (not seqprt-style) mnemonic or text statement, or a list of one or more mnemonic and text statements in any combination.

All heading items are printed on each page at the given row and column position. Non-heading items are printed on lines following the last heading item on the page.

For mnemonic-data headings, normally only the mnemonic name and/or units would be referenced in the *mnemonic-stmnt*.

4.8 Newfile Statement

Syntax

`newfile` *interval* [, [*time*] [, *action*]]

Discussion

Use the `newfile` statement to periodically open a new output file, rather than collecting all output in one ever-growing file. A new file is opened each time the sequential print writes a line of output and *interval* time has elapsed since the current file was opened. The *interval* may be given in seconds or as a time specification in the form '`hh:mm:ss`'.

Each time the interval expires and the seqprt goes to write a line of output, it closes the current output file and renames it so as to add a date stamp suffix in the form '`yyyyjjjhhmmss`' (4-digit year; 3-digit julian day; and 2-digit hours, minutes, and seconds). Then the seqprt opens a new file with the given output file name and restarts the interval timer.

If *time* is given, the first new file is opened the first time seqprt writes and time is past that time of the current day, or the next day if the time already is past, and then on the given *interval*. The *time* may be given in seconds past midnight or as a time specification in the form '`hh:mm:ss`'.

The *action* may be any UNIX command. It is executed each time `newfile` timer expires, after the output file has been closed and renamed, but before the new output file is opened. The new name of the just closed and renamed file is written to the standard input of the *action* process.

Note that if the action creates a file that is the seqprt output file, and the seqprt is set to append to it's output file, it will append to this new file; else it will truncate and overwrite this file.

5 Format Strings

Page and seqprt definition mnemonic statements contain format strings detailing how mnemonic information is to be printed. The format string contains text, spaces, and special colon-delimited *keys*.

Each key corresponds to a piece of information about a mnemonic, such as it's name or value, and that information is printed on the page in place of the key. Text and spaces are printed as given.

Recognized keys are:

key	alias	information referenced
`:name:`	`:n: :mnem:`	name string from database
`:value:`	`:v:`	value
`:units:`	`:u:`	units-of-measure string from database
`:static:`	`:s:`	static flag state
`:quality:`	`:q:`	qualify flag state
`:limits:`	`:l:`	limits flag state
`:rail:`	`:r:`	rail limit flag state
`:delta:`	`:d:`	delta limit flag state*

(*) are unimplemented

On a page, each key is replaced by the data associated with it. Each key may contain an optional C-language, `printf()`-style *conversion specification* (see below for details) to control how the data is displayed. (This should not be confused with the conversion field in the mnemonic statement.)

The display software includes a set of default conversion specifications for each key. If the page designer supplies a conversion specification with the key, it overrides the default. Conversion specifications are to be appended to the key text as the following list of defaults illustrates:

`:name%-8s:`	8-character string, left justified
`:value%8x:`	8-digit hex number, right justified
`:units%-4s:`	4-character string, left justified
`:static%1s:`	1-character string
`:quality%1s:`	1-character string
`:limits%2s:`	2-character string
`:rail%1s:`	1-character string
`:delta%1s:`	1-character string

Flag values are displayed as follows:

`static`	'`*`'	static, else blank
`quality`	'`Q`'	bad quality, else blank
`limits`	'`RL`'	red low violation

	'YL'	yellow low violation
	'YH'	yellow high violation
	'RH'	red high violation
rail	'H'	rail high
	'L'	rail low
delta	'D'	delta limit violation, else blank

Typically, conversion specifications are used only with the :value: key. Obviously, it makes little sense to supply conversion specifications for mnemonic flags, since the information printed cannot be changed by the user.

Spaces and other characters between keys are preserved on the display. The string :: is displayed as ':'. Newline characters are not allowed in format strings.

If no value has been received for a telemetry item, the string 'NV' is displayed in the value field. The string produced from a given format is displayed in white text on a red background if the associated mnemonic is in red limit violation, in black on yellow if it is in yellow violation, and in grey on the default background if it is static.

Note that pure text items which may be adjacent to mnemonic items will not change color for violations or static conditions. Consider the following page definition fragment:

```
( 1,   1, "Heater #1 temperature: ")
mnem_1 ( =, +3, " :value%6.3f: ")
mnem_2  ( =, +3, "Heater #2 temperature::   :value%6.3f:")
```

For mnemonic 'mnem_1', only the value changes color on violations and static. For mnemonic 'mnem_2', the entire string changes color.

Here are some example format strings and the output they might produce.

```
" :n:    :v:    :u:    :s: :q: :l:"
MNEM_2 23d4   CTS   *QRL

"Heater voltage (:n%-6s:)::   :v%5.2f:   :u:   :s: :q: :l: "
Heater voltage (MNEM_1):  23.21 V   *QYH
```

5.1 Format Conversion Specifications

The following is adapted from the FreeBSD 2.2.7 version of the 'printf' manpage

Each conversion specification is introduced by the character %. The mnemonic value (before or after discrete or analog conversion) with the conversion specifier. After the %, the following appear in sequence:

- Zero or more of the following flags:
  - A # character specifying that the value should be converted to an "alternate form". For c, d, i, n, p, s, and u, conversions, this option has no effect. For o conversions, the precision of the number is increased to force the first character of the output string to a zero (except if a zero value is printed with an explicit precision of zero). For x and X conversions, a non-zero result has the string '0x' (or '0X' for X conversions) prepended to it. For e, E, f, g, and G, conversions, the result will always contain a decimal point, even if no digits follow it (normally, a

decimal point appears in the results of those conversions only if a digit follows). For g and G conversions, trailing zeros are not removed from the result as they would otherwise be.

- A zero '0' character specifying zero padding. For all conversions except n, the converted value is padded on the left with zeros rather than blanks. If a precision is given with a numeric conversion (d, i, o, u, i, x, and X), the '0' flag is ignored.
- A negative field width flag '-' indicates the converted value is to be left adjusted on the field boundary. Except for n conversions, the converted value is padded on the right with blanks, rather than on the left with blanks or zeros. A '-' overrides a '0' if both are given.
- A space, specifying that a blank should be left before a positive number produced by a signed conversion (d, e, E, f, g, G, or i).
- A '+' character specifying that a sign always be placed before a number produced by a signed conversion. A '+' overrides a space if both are used.

- An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the left-adjustment flag has been given) to fill out the field width.
- An optional precision, in the form of a period '.' followed by an optional digit string. If the digit string is omitted, the precision is taken as zero. This gives the minimum number of digits to appear for d, i, o, u, x, and X conversions, the number of digits to appear after the decimal-point for e, E, and f conversions, the maximum number of significant digits for g and G conversions, or the maximum number of characters to be printed from a string for s conversions.

  The conversion specifiers and their meanings are:

  `diouxXb` The int (or appropriate variant) argument is converted to signed decimal (`d` and `i`), unsigned octal (`o`), unsigned decimal (`u`), unsigned hexadecimal (`x` and `X`), or unsigned binary (`b`) notation. The letters '`abcdef`' are used for `x` conversions; the letters '`ABCDEF`' are used for `X` conversions. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros.

  `eE` The double argument is rounded and converted in the style [-]d.ddde+-dd where there is one digit before the decimal-point character and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero, no decimal-point character appears. An E conversion uses the letter `E` (rather than `e`) to introduce the exponent. The exponent always contains at least two digits; if the value is zero, the exponent is 00.

  `f` The double argument is rounded and converted to decimal notation in the style [-]ddd.ddd, where the number of digits after the decimal-point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is explicitly zero, no decimal-point character appears. If a decimal point appears, at least one digit appears before it.

`g` The double argument is converted in style f or e (or E for G conversions). The precision specifies the number of significant digits. If the precision is missing, 6 digits are given; if the precision is zero, it is treated as 1. Style e is used if the exponent from its conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional part of the result; a decimal point appears only if it is followed by at least one digit.

`c` The int argument is converted to an unsigned char, and the resulting character is written.

`s` The "char *" argument is expected to be a pointer to an array of character type (pointer to a string). Characters from the array are written up to (but not including) a terminating NUL character; if a precision is specified, no more than the number specified are written. If a precision is given, no null character need be present; if the precision is not specified, or is greater than the size of the array, the array must contain a terminating NUL character.

`T` The time mnemonic value is converted to ddd-hh:mm:ss.ffffff notation.

`D` The date mnemonic value is conerted to yy-ddd-hh:mm:ss.ffffff notation.

5.1.1 Complex Format Conversions

The `T` and `D` format conversion specifications deal with complex data objects and so are subject to special handling. This allows the standard conversions to be modified, which can be especially useful with time and date values.

These specifications actually are shorthand for more complex specifications of the form '`%`*w*`[`*e*`]`*S*', where *w* is the overall field width, *e* is a special format specification, and *S* is the conversion character.

The following table gives the longer format for which each special format conversion character is shorthand.

`T` ↦ `%10[%S.%06f]T`

`D` ↦ `%22[%y-%j-%T.%f]D`

You can use this type of complex format specification in your page and sequential print definitions to gain extra control on how the complex data types are displayed. Here are a few examples:

`:v%17[%s.%f]D:`
: will display a date as seconds and microseconds since the epoch, such as '`1098821013.412374`'.

`:v%12[%b %e, %Y]D:`
: will display a date in the form '`Oct 12, 2004`'.

`:v%13[%D]T:`
: will display a time as seconds, minutes:seconds, hours:minutes:seconds, or days-hours:minutes:seconds, as needed depending on the magnitude of the iinputs time. So if the mnemonic has a value of 86420, it will be displayed as '`1-00:00:20`', while a value of 3003 comes out as '`50:03`'.

5.1.1.1 Time Formats

The following is a complete listing of conversions available in formatting time mnemonic values within the `%T` conversion.

Note that some conversions may be *extended* by including a '-' sign immediately after the % in the conversion spec. See the individual conversion character below for an explanaation of what extending will do in each case where it is available. In general it allows you to give a base format and let the formatter fill in where it needs to.

`d` is replaced by the number of days spanned by the input time.

`D` is replaced by the number of days spanned by the input time. When extended, operates as follows:

if input time is	then output format is
< 60	%-S
>= 60, < 3600	%-M:%s
>= 3600, < 86400	%-H:%m:%s
>= 86400	%-D-%h:%m:%s

If zero fill was called for in the'`%D`', it will be called for in the extended conversions, and a zero field will be printed for each element in '`%-0D-%02h:%02m:%02s`'

`h` is replaced by the number of hours spanned by the input time, 0-23. If more than 23 hours were spanned, the number produced will be the remainder of hours when the input time is divided into days.

`H` is replaced by the total number of hours spanned by the input time. When extended, operates as follows:

if input time is	then output format is
< 60	%-S
>= 60, < 3600	%-M:%s
> 3600	%-H:%m:%s

If zero fill was called for in the'`%H`', it will be called for in the extended conversions, and a zero field will be printed for each element in '`%-0H:%02m:%02s`'.

`m` is replaced by the number of minutes spanned by the input time, 0-59. If more than 59 minutes were spanned, the number produced will be the remainder of minutes when the input time is divided into hours.

`M` is replaced by the total number of minutes spanned by the input time. When extended, operates as follows:

if input time is	then output format is
< 60	%-S
> 60	%-M:%s

If zero fill was called for in the'`%M`', it will be called for in the extended conversions, and a zero field will be printed for each element in '`%-0M:%02s`'.

`Ss` is replaced by the number of seconds in the input time. The `s` will convert only seconds up to 59. If more than 59 seconds were input, the number produced will be the remainder of seconds when the input time is divided into minutes.

`Ff` is replaced by the microseconds component of the input time. If `F` is specified and the microseconds component are zero, no conversion is done. If `F` does result in a conversion, the decimal point (‘`,`’) is added as part of the conversion and should not be given separately.

`T` is shorthand for ‘`%03d-%02h:%02m:%02s.%06f`’.

`%` is replaced by ‘`%`’. This is nonsense.

5.1.1.2 Date Formats

The following is a complete list of conversions available in formatting date mnemonic values within the code%D conversion.

`Aa` is replaced by the weekday name. `A` produces the full name; `a` produces the abbreviated name.

`Bbh` is replaced by the month name. `B` produces the full name; `b` and `h` produce the abbreviated name.

`C` shorthand for ‘`%a %b %e %H:%M:%S %Y`’.

`c` shorthand for ‘`%m/%d/%y %H:%M:%S`’.

`D` shorthand for ‘`%m/%d/%y`’.

`de` is replaced by the 2-digit day of the month, 1 to 31. `d` preceeds single digits with ‘`0`’; `e` with space.

`f` is replaced by the microseconds field of the input time as a 6-digit, zero-filled decimal number.

`Hk` is replaced by the hour on a 24-hour scale, 0-23. `H` preceeds single digits with ‘`0`’; codek with space.

`Il` is replaced by the hour on a 12-hour scale, 1-12. `I` preceeds single digits with ‘`0`’; `l` with space.

`Jj` is replaced by the day of the year (Julian day). `J` begins with day ‘`000`’; `j` with day ‘`001`’.

`M` is replaced by the minute, ‘`00`’-‘`59`’.

`m` is replaced by the month, ‘`01`’-‘`12`’.

`n` is replaced by a newline, which cannot appear on a page, but might be useful to an ITOS programmer.

`p` is replaced by ‘`AM`’ or ‘`PM`’, as appropriate.

`R` is shorthand for ‘`%H:%M`’.

`r` is shorthand for ‘`%I:%M:%S %p`’.

`S` is replaced by the number of seconds, ‘`00`’-‘`59`’.

`s` is replaced by the number of seconds in the input time, which are seconds from the UNIX epoch in local time.

`TX` is shorthand for '`%H:%M:%S`'.

`t` is replaced by a tab character, which is not useful in page definitions.

`UW` is replaced by the week number of the year, '`00`'-'`53`'. `U` takes Sunday as the first day of the week; `W` takes Monday.

`w` is replaced by the day of the week, '`0`'-'`6`', with Sunday as day '`0`'.

`x` is shorthand for '`%m/%d/%y`'.

`Yy` is replaced by the year. `y` produces the 2-digit year, '`00`'-'`99`'; '`Y`' the four-digit year.

`Z` is replaced by the timezone abbreviation.

6 Page Programs

Programs written for the X Window System generally employ *resources* to set various parameters needed by the program, and many of these are configurable by the user.

Here is a list of resources used by the ITOS all page programs:

Name	Type	Default	Explanation
`pageName`	String	empty string	Page name.
`interval`	int (milliseconds)	0	Page update interval.
`displaySlot`	int	0	Display slot in which to place the page.
`baseFlag`	Boolean	false	If true, this is a base page.
`ySlotOffset`	Position (integer)	0	Y offset of slots 3 & 4 from the top of the screen.
`pageListIndex`	int	NULL_INDEX (-1)	Index to this page's entry in the ITOS page list.
`font`	XFontStruct	XtDefaultFont	The normal font for all page items.
`font1`	XFontStruct	XtDefaultFont	The smaller font for all page items.
`font2`	XFontStruct	XtDefaultFont	The bigger font for all page items.
`font3`	XFontStruct	XtDefaultFont	The biggest font for all page items.
`pageDefFile`	String	empty string	Name of the page definition file.
`snap`	Boolean	false	If true, snap the page instead of displaying it.
`outputFile`	String	NULL	Name of snap output file.
`append`	Boolean	false	If true, append snap output to output file.

The following is the list of resources supported by the telemetry page program, `dsp_mnepage`, and the packet counter page program, `dsp_pktcount`:

Name	Type	Default	Explanation
`itemBorderWidth`	Dimension	1	Border width of each page item.
`itemInternalHeight`	Dimension	2	Internal margin for each page item.
`pageMargin`	Dimension	7	Page outer margins.
`lineSpace`	Dimension	4	Space between each line on the page.
`numRows`	Dimension	15	Number of rows on the page.
`numCols`	Dimension	80	Number of columns on the page.
`discreteColor0`	Pixel	black	Color for color code 0.
`discreteColor1`	Pixel	red	Color for color code 1.
`discreteColor2`	Pixel	green	Color for color code 2.
`discreteColor2`	Pixel	yellow	Color for color code 3.
`discreteColor4`	Pixel	blue	Color for color code 4.
`discreteColor5`	Pixel	magenta	Color for color code 5.
`discreteColor6`	Pixel	cyan	Color for color code 6.
`discreteColor7`	Pixel	white	Color for color code 7.

These are the resources for the telemetry page snap program, `dsp_mnepage_snap`:

Name	Type	Default	Explanation
`pageName`	String	"testpage"	Page name.
`interval`	int	0	Snap update interval.
`outputFile`	String	NULL	Name of snap output file.
`append`	Boolean	false	If true, append snap output to output file.
`pageListIndex`	int	NULL_INDEX (-1)	Index to this page's entry in the ITOS page list.
`pageDefFile`	String	empty string	Name of the page definition file.

7 Seqprt Program

Name	Type	Default	Explanation
`pageName`	String	empty string	Page name.
`window`	Boolean	false	If true, output to an X window.
`events`	Boolean	false	If true, output to the events subsystem.
`printer`	String	NULL	If non-NULL, name of the printer to which to output.
`file`	String	NULL	If non-NULL, name of the file to which to output.
`append`	Boolean	false	If true, append to the output file.
`outputDir`	String	NULL	Name of the directory where the output file is created. If NULL, this is set from the global mnemonic `gbl_sprtdir`. If the global is not present, it defaults to '`./prints`'.
`font`	XFontStruct	XtDefaultFont	The normal font for all page items.
`font1`	XFontStruct	XtDefaultFont	The smaller font for all page items.
`font2`	XFontStruct	XtDefaultFont	The bigger font for all page items.
`font3`	XFontStruct	XtDefaultFont	The biggest font for all page items.
`pageListIndex`	int	NULL_INDEX (-1)	Index to this page's entry in the ITOS page list.
`lineLength`	int	2048	Character column at which long lines are broken.
`linesToSave`	int	256	Number of lines through which the user can scroll the display.
`interval`	int (milliseconds)	0	Page update interval.
`seqprtDefFile`	String	NULL	Name of the sequential print definition file.

Table of Contents